Adaptive Power Control for Sober High-Performance Computing

Ismail Hawila,[1] Sophie Cerf,[2] Raphaël Bleuse,[1] Swann Perarnau,[3] and Éric Rutten[1]

***Abstract*— Soberness—in terms of electrical power—of data centers and high-performance computing systems is becoming an important design issue, as the global energy consumption of information technologies is rising at considerable levels. This issue is all the more complex as these systems are increasingly heterogeneous and variable in their behavior, for example, w.r.t. performance and power consumption, and less predictable, thus demanding runtime management and feedback control.**

This paper addresses the problem of the control of the power allocated to processors and hence their energy consumption and performance. The use of feedback control allows the energy consumption to be reduced by decreasing the speed without losing performance, by exploiting periods where read/write operations slow the progress. Previous works present limitations regarding both modeling (nonlinear models with numerous parameters) and control performance (mainly instability caused by platform variations). We develop a novel adaptive control that is robust to the variety of execution platforms while maintaining the existing global goals of energy management. We evaluate—on a real system using the Grid'5000 testbed—the robustness of the control to changes in initial parameters and to disturbances, and we compare it with the previous proportional-integral (PI) control. Our adaptive control approach allows for up to 25% energy savings.

Keywords: Adaptive Control, Control for Computing, Energy Consumption

I. Introduction

A. Control Theory for High-Performance Computing

The study of complex systems (e.g., climate, genomics, high-energy physics) requires powerful computing infrastructures in order to run codes of compute-intensive and data-intensive applications. Dedicated computing platforms, referred as "high-performance computing" (HPC) systems, are built to address the needs of such studies. Both the hardware and software stacks of HPC systems are getting increasingly complex to cope with the increasing computing requirements within a finite and economically tractable power budget. This complexity is expressed, for example, by the increased failure rate of hardware, varying and data-dependent behaviors of applications, complex and hard-to-predict interactions between running applications.

The energy consumption of these computing systems is becoming an important problem, and hence there is a need for more soberness through regulation of energy and power. The objective of power management is to have optimal performance under a given power budget. In order to respond to changes in system and application behavior, the power management must be dynamic and requires measurement of the online performance of the running application, with decisions taken at runtime. The need for feedback regulation starts to emerge.

An approach to such problems, termed autonomic computing, was proposed in [1]. In autonomic computing, systems are self-adaptive; that is, they manage themselves according to given goals and objectives, with a notion of feedback loop involving a variety of decision mechanisms such as ad hoc implementations, artificial intelligence and machine learning, scheduling, and constraint programming. A particularly interesting approach involves control theory [2], [3], [4].

Control theory has been used in computing systems, especially in cloud systems [5], [6], and in real-time systems [7] and the Internet of Things [8]. Its use for HPC systems is recent, however. For example, Yabo *et al.* [9] used a control theory approach for management of scientific workflows in HPC systems. Recent works on energy management of computing systems use control theory. Imes *et al.* [10] used control theory for energy minimization in real-time systems, where they rely on the use of a dynamic voltage frequency scaling actuator to enforce a power limit. Other works use the Running Average Power Limit (RAPL) [11] mechanism as an actuator for power regulation: this actuator allows the enforcement of a maximum power level for processors' consumption, called the *powercap*. For example, Imes *et al.* used RAPL combined with a control-theory approach on real-time systems to control power while specifying a performance goal [12].

The use of control theory for energy management in HPC systems does not have a lot of presence in the literature. Therefore, when tackling power regulation for HPC systems, having an accurate measure of applications performance is crucial. For this, Ramesh *et al.* [13] were able—based on interviews with HPC application specialists—to define the performance of applications as a progress metric that correlates with the scientific usefulness and found that for some applications the progress correlates with the applied powercap.

B. Contributions

The present work builds on and improves on the energy regulation for sober HPC systems proposed by [14]. That previous work defined a nonlinear model with first-order dynamics and a proportional-integral (PI) controller relying on RAPL as a power actuator to decrease energy consumption while sustaining an appropriate level of performance. However, the work was not robust enough to disturbances

[1]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France {ismail.hawila, raphael.bleuse, eric.rutten}@inria.fr

[2]Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France sophie.cerf@inria.fr

[3]Argonne National Laboratory, Lemont, IL, USA swann@anl.gov

and changes in the experiment environment. In particular it relied on a difficult-to-tune model with many parameters and did not answer the need for adaptation of the controller to various execution environments (facing the variability of processors) and time variations (concurrency on the platform, aging, temperature, etc.). To tackle these issues, we design a new adaptive controller that is more robust than the PI and simpler to design. The contributions of this work are as follows:

- Identification of the limitation of the model and the available control in [14]
- Design of an adaptive controller that is robust to different clusters and to the applications environment
- A novel initialization method called `two-runs`, well suited for short and repeated controller executions
- Extensive experimental evaluation of the proposed controller on the data center testbed.

The remainder of this paper is organized as follows. Section II presents the system and describes the problem and the control formulation. Section III reviews the previous work and outlines its limits. Section IV presents the adaptive controller and `two-runs`, its initialization strategy, which are then validated and discussed in Section V. Section VI summarizes the conclusions and briefly presents perspectives for future work.

II. System Description

This section pedagogically presents the HPC application and architecture of our proposed system from a computing perspective (Section II-A) and then translates it in a proper control formulation (Section II-B) highlighting the plant, actuator, and sensor. An analysis of the system's challenges (Section II-C) ends this section.

A. *HPC System: Application and Architecture*

This work aims at regulating the performance and energy consumption of an HPC application using powercap actuation. The performance of an application can be analyzed by determining which resource is a limiting factor. We distinguish three major limiting resources: *(i)* computation, *(ii)* memory, and *(iii)* data exchanges with input/output devices. An application is said to be compute-bound (memory-, I/O-, resp.) when the time to execute the application is determined by the time spent computing (communicating with memory, waiting on I/O operations, resp.). Complex applications can be broken down into sequences of phases with different boundedness.

Modern processors are able to adapt the speed of computation to save energy. The original approach described in [14] and refined in this work takes root in the observation that during non-compute-bound phases, processors can be slowed down without affecting the performance of the application.

A typical HPC system architecture is as follows. The infrastructure is organized as clusters of computing resources with identical hardware specifications. A cluster is a set of machines called nodes. Each node contains one or several sockets—usually 1, 2, or 4. Each socket hosts a single processor. We use the words socket or processor interchangeably. Refer to Section V-A for details on the clusters (`gros`, `dahu`, `chifflot`) used in this work. Modern Intel processors implement RAPL (Running Average Power Limit) [11], which allows the user to define a maximum power draw per socket, or powercap. This power limit is enforced by RAPL through an internal loop that acts on processors' states.

HPC applications are executed on a subset of the HPC infrastructure. For the sake of simplicity, we consider in this work an application executing on a single node. On each node, a dedicated software component is in charge of centralizing the application sensing and the hardware actuation. Here this role is assumed by the Node Resource Manager (NRM) middleware [15]. NRM hosts and runs the controller. Figure 1 depicts the computing architectural view of the system, where the signals used in the control loop are represented by the arrows.

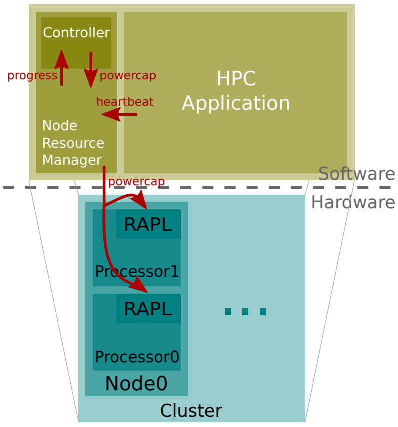

Fig. 1: Architecture of the system: example of a cluster consisting of a node with two processors. Arrows represent the signals used in the control loop (see Fig. 2).

B. *Control Formulation*

From a control theory perspective, the HPC application along with the hardware is considered as the plant. We measure the performance of this application at runtime through a progress sensor. The RAPL actuator allows one to dynamically vary the power available for the application. Note that the internal process of the RAPL mechanism—undisclosed by the manufacturer—is not considered in this work; hence we adopt a black-box approach. The user sets their objective as an allowed performance degradation with respect to the performance of a full-power execution. The controller computes the control signal—here the powercap—based on the progress measure and the translated user reference. The control loop is shown in Fig. 2.

The control signal is the powercap, denoted $u(t_i)$. The power actuator (RAPL) guarantees that a given average power is maintained by modifying the processor internal state. The sampling time $\Delta_t = t_i - t_{i-1}$ is defined by the frequency with which we update the powercap.

The performance measurement is collected by NRM by instrumenting the HPC application with a lightweight library

[13]. This instrumentation sends a message reporting the amount of progress performed since the last message, which is derived as a heartbeat. The NRM sensor outputs a progress signal, a smoothed version of the heartbeat signal defined as the median of the heartbeat's arrival frequency between t_i and the last sampling period t_{i-1}:

$$y(t_i) = \underset{\forall k,\, t_k \in [t_{i-1}, t_i[}{\text{median}} \left(\frac{1}{t_k - t_{k-1}} \right). \tag{1}$$

C. System Analysis

We are now interested in analyzing the control problem regarding possible disturbances, noises, and nonlinearities. First, the literature emphasized that the RAPL actuator is not accurate [16], and this error increases as the control signal grows [14]. Figure 3 gives the static characteristic of a specific HPC application [14]. Overall, the more power given to the application, the higher its progress will be. The system gain varies with the cluster and notably depends on the specifications of the processors. The system presents a nonlinearity: variations of power around low values have a significant impact on progress, whereas variations around large values of power only slightly improve the output. Such behavior is explained by the memory boundedness of the application. Although not visible on this modeling, the system additionally presents noise in the sensing, as well as the presence of outlier values. While the exact cause remains uncertain, the sensing quality is observed to decrease with the number of processors. More details on those variabilities are given in Section III-B.

III. Background

This section presents the state of existing work regarding modeling and control of HPC power regulation [14]. Their limitations are then discussed and illustrated.

A. Existing Models and Control

A study of the static case is first presented. Identification from data collected experimentally (similar to Fig. 3) leads to the formulation of a static model as an asymptotic regression model:

$$y = K\left(1 - e^{-\alpha(\beta \cdot u + \gamma - \delta)}\right) \tag{2}$$

in which β and δ are parameters accounting for the RAPL actuator inaccuracies. All parameters are cluster specific, as presented in Table I. The nonlinearity of the system (highlighted in Section II-C) is modeled by the exponential function. A change of notation allows one to cope with this nonlinearity. The new input and output signals are formulated as follows:

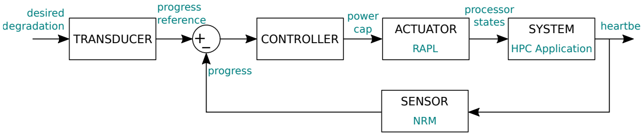

Fig. 2: Control feedback loop, abstraction of the architecture.

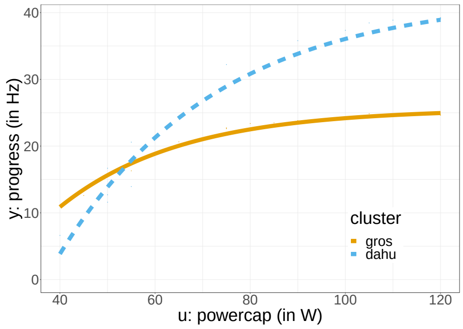

Fig. 3: Static characteristic modeling of the system revealing cluster-related variations and nonlinearities [14].

$$\begin{aligned} \mathbf{u} &= -e^{-\alpha(\beta \cdot u + \gamma - \delta)} \\ \mathbf{y} &= y - K. \end{aligned} \tag{3}$$

One can translate the static characteristic equation Eq. (2) as

$$\mathbf{y} = K\mathbf{u}. \tag{4}$$

In a second step, the dynamical behavior is modeled. A first-order model is used as a trade-off between simplicity and accuracy:

$$\mathbf{H}(s) = \frac{\mathbf{Y}(s)}{\mathbf{U}(s)} = \frac{K}{1 + \tau s}, \tag{5}$$

where τ is the time constant characterizing the transient behavior; linked with the dynamics of the sensor; see Eq. (1). Since our system is in discrete time with nonconstant sampling time, Eq. (5) can be rewritten as

$$\mathbf{y}(t_{i+1}) = \frac{K\Delta t_{i+1}}{\Delta t_{i+1} + \tau}\mathbf{u}(t_i) + \frac{\tau}{\Delta t_{i+1} + \tau}\mathbf{y}(t_i), \tag{6}$$

with $\Delta t_i = t_i - t_{i-1}$.

For the regulation, a PI controller has been developed. The objective is given in terms of a degradation factor ϵ. The reference setpoint is computed by using this degradation factor

$$y_{\text{ref}} = (1 - \epsilon) \cdot y_{\max}, \tag{7}$$

where $y_{\max}$ is the nominal progress computed by using Eq. (2), where the control input is set to the maximum power of the cluster $u_{\max}$. With the error signal being $e(t_i) = y_{\text{ref}} - y(t_i)$, the PI control formulation is

$$\text{PI}(s) = K_P + \frac{K_I}{s}. \tag{8}$$

That is rewritten in discrete time with nonconstant sampling time as:

$$\mathbf{u}(t_i) = (K_I \Delta t_i + K_P) \cdot e(t_i) - K_P \cdot e(t_{i-1}) + \mathbf{u}(t_{i-1}). \tag{9}$$

The gains K_P and K_I are set by pole placement as $K_P = \frac{\tau}{K\tau^*}$ and $K_I = \frac{1}{K\tau^*}$, respectively, where $\tau^* = 10\,\text{s}$ defines the desired dynamical behavior of the controlled system. The PI design and its performance are presented in detail in [14].

Description	Notation	Unit	gros	chifflot	dahu
RAPL slope	β	[1]	0.83	1.03	0.94
RAPL offset	δ	[W]	7.07	4.04	0.17
	α	[W^{-1}]	0.047	0.028	0.032
power offset	γ	[W]	28.5	37.04	34.8
linear gain	K	[Hz]	25.6	42.82	42.4
time constant	τ	[s]	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

TABLE I: Model and controller parameters per cluster [14]

B. On the Need for Robustness

An HPC application such as our system undergoes many variations of its behavior, depending on *(i)* the cluster, *(ii)* the node, *(iii)* the run, and even *(iv)* during the runtime. This section illustrates and analyzes those variations and highlights the limitation of the state-of-the-art control, motivating our adaptive control approach.

Open-loop experiments were conducted to illustrate the system's variations, as presented in Fig. 4. The input is a stepped signal of period 20 s. First, we observe that depending on the cluster, the system gain is significantly different. There is about a factor 2 between the `gros` and `dahu` clusters. This is due to the different characteristics of the nodes composing the cluster; see Table II. Second, the effective node on which the application is launched also affects the system gain, as can be seen comparing data from `dahu-a` and `dahu-b` in Fig. 4. Third, the *run* also creates variability. Between two repetitions of the same experimental conditions, results vary, as presented in Fig. 4 for data `dahu-a run-1` and `dahu-a run-2`. Eventually, even during the same execution we can observe variations, as can be seen around 25 s for `dahu-a run-1`.

Regarding the closed loop results, we observe that a PI control such as from the state-of-the-art control [14] can lead to system instability, as illustrated in Fig. 5 (right plot). The measured performance signal oscillates around the reference due to oscillations of the control signal. Note the specificity of our system for which the experiment stops when the application finishes its execution: there is no longer a system, hence no behavior to show after 150 s. The PI also is not robust to the presence of outlier measures. Figure 5 (left plot) illustrates a run where the control signal presents peaks and large oscillations due to the presence of outliers. Note that those experiments are selected to highlight the worst-case behavior of the PI.

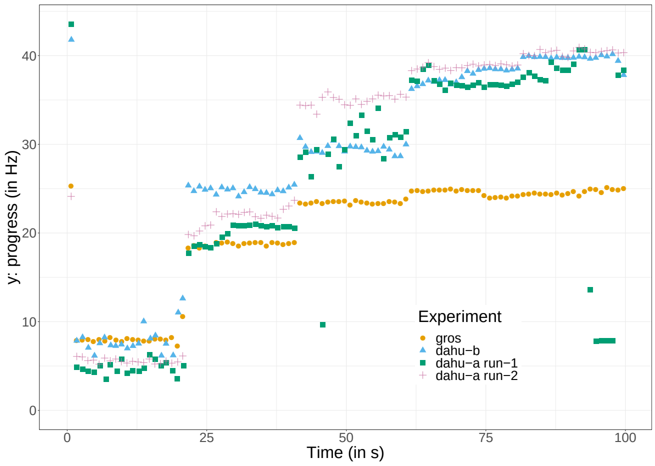

Fig. 4: Open-loop behavior for several experimental conditions, highlighting the variabilities in the system and hence the need for robustness. The control input (u, the powercap) is a stepped signal taking the values $[40, 60, 80, 100, 120]$ W at times $[0, 20, 40, 60, 80]$ s.

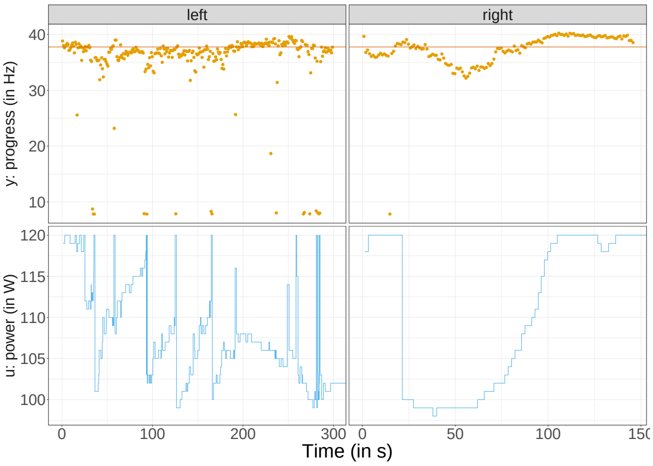

Fig. 5: A feedback controller with constant parameters is affected by outliers and can drive the system to instability. `dahu` cluster. Reference is set with $\epsilon = 0.03$ (red line).

Those analyses motivate the need for a control robust to variations of the machines, the run, and at runtime. While the runtime variation is a classic motivation for adaptive control, robustness to static environmental conditions are less usual. The nature of our computing system (finite time and short executions, largely repetitive) allows us to consider this renewed contribution of adaptive control.

IV. Adaptive Power Control

The core idea of adaptive control is to cope with changes in the system through updates of the controller. An adaptation feedback is used—in addition to the control loop—as illustrated in color in Fig. 6. The adaptation aims to update the controller parameters to follow a model-reference online using estimation methods. We use discrete Model-Reference Adaptive Control [17], presented in Section IV-A and translated to our use case in Section IV-B. Section IV-C presents `two-runs`, a 2-step initialization process for the adaptation parameters.

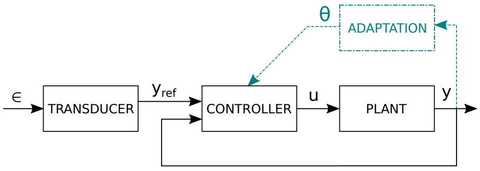

Fig. 6: Adaptive control of HPC application power.

A. Discrete Model-Reference Adaptive Control

We present MRAC where the parameter estimation is based on a projection algorithm. Proofs of Lyapunov stability are given in [17]. We consider a single-input, single-output system represented as follows:

$$A(q)y(k) = B(q)u(k), \tag{10}$$

where q is the forward shift operator and A and B are polynomials defined as

$$\begin{aligned} A(q) &= q^n + a_1 q^{n-1} + \ldots + a_n \\ B(q) &= b_0 q^m + b_1 q^{m-1} + \ldots + b_m. \end{aligned} \tag{11}$$

We define the delay $d = \deg(A) - \deg(B) = n - m$.

The control law is defined based on model matching:

$$\begin{aligned} u(k) &= \frac{T(q)}{R(q)} u_c(k) - \frac{S(q)}{R(q)} y(k) \\ y_m(k) &= \frac{B_m(q)}{A_m(q)} u_c(k), \end{aligned} \tag{12}$$

where u_c is the command signal; y_m is the model reference to be achieved by the output y; R, S, and T are the control polynomials; and A_m and B_m represent the reference model. The reference model should be carefully chosen to ensure stability, controllability, and robustness [18], [19].

The filtered output signal—linear estimation model in terms of the controller—is defined as

$$y_f(k+d) = b_0 u(k) + \phi^T(k)\theta, \tag{13}$$

where θ is the parameter vector containing the parameters of $R(q)$ and $S(q)$:

$$\theta = [r_1, \ldots, r_{n-1}, s_0, \ldots, s_{n-1}]^T \tag{14}$$

and ϕ is the regression vector defined as

$$\phi(k) = [u(k-1), \ldots, u(k-n+1), y(k), \ldots, y(k-n+1)]^T . \tag{15}$$

Then the control law can be written as

$$u(k) = -\frac{1}{b_0}\left[\phi^T(k)\hat{\theta}(k) - q^{-n+1} B_m u_c(k)\right]. \tag{16}$$

The expression of the parameters vector estimate $\hat{\theta}$ is found by using the projection algorithm to minimize a given cost function [20]:

$$\begin{aligned} \hat{\theta}(k) = \hat{\theta}(k-1) + \frac{1}{\phi^T(k-d)\phi(k-d)} \phi(k-d) \, [y_f(k) \\ - b_0 u(k-d) - \phi^T(k-d)\hat{\theta}(k-1)\Big]. \end{aligned} \tag{17}$$

B. Application to Power Regulation

Our power-to-progress system can be expressed from Eq. (6) as

$$y(k+1) = b_0 u(k) + a_0 y(k). \tag{18}$$

We have $(n, m, d) = (1, 0, 1)$. The reference to be followed is expressed by $y_{\mathrm{m}} = u_c$. The parameter vector and the regressor are

$$\theta = [s_0] \tag{19}$$

$$\phi(k) = [y(k)], \tag{20}$$

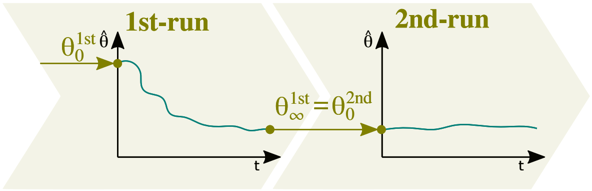

Fig. 7: Schematic representation of the `two-runs` initialization process.

Which allows one to write Eq. (18) as

$$y(k+1) = b_0 u(k) + \phi(k)^T \theta. \tag{21}$$

The control law along with the parameter update used is

$$u(k) = -\frac{1}{b_0}\left[\phi^T(k)\hat{\theta}(k) - b_m y_{\text{ref}}\right] \tag{22}$$

$$\begin{aligned} \hat{\theta}(k) = \hat{\theta}(k-1) + \frac{1}{\phi^T(k-1)\phi(k-1)} [a_m y(k-1) \\ - b_0 u(k-1) - \phi^T(k-1)\hat{\theta}(k-1)\Big] \, \phi(k-1). \end{aligned} \tag{23}$$

C. `two-runs` Initialization

The parameter vector estimate $\hat{\theta}$ needs an initialization, denoted θ_0. We present `two-runs`—a two-step initialization process that takes advantage of our computing application: a run is cheap, and ill-tuned control does not damage the system. `two-runs` works as follows. A first run (*1st-run*) is launched with the theoretical—yet general and unfitted—value for θ guessed from the steady-state value. Then (*2nd-run*) the converged value of the parameter estimate vector of *1st-run* is used for the subsequent initializations. By *run* we mean here an execution of the application on the experimental platform. We assume that the adaptation has converged by the end of the first run. This hypothesis is realistic given the large duration of an application execution regarding the adaptation's dynamics. A schematic representation of `two-runs` is given in Fig. 7.

$$\theta_0^{\text{1st}} = a_m - \frac{b_0}{K} \qquad (\textit{1st-run}) \tag{24}$$

$$\theta_0^{\text{2nd}} = \theta_\infty^{\text{1st}} \qquad (\textit{2nd-run}) \tag{25}$$

We find the value of θ_0^{1st} as the solution of Eq. (23) in steady state:

$$\theta = \theta + \frac{1}{\phi^T\phi}\phi\left[a_m y - b_0 u - \phi^T\theta\right]. \tag{26}$$

From Eq. (20) with a one-dimensional parameter vector, one has $\phi = y = \phi^T$, and thus

$$\begin{aligned} 0 &= \frac{1}{y}\left[a_m y - b_0 u - y\theta\right] \\ \theta &= a_m - b_0 \frac{u}{y}. \end{aligned} \tag{27}$$

Equation (25) derives from the latter equation combined with Eq. (4). The regressor vector is initialized by using Eq. (4):

$$\phi(0) = K\mathbf{u}(0) + K. \tag{28}$$

V. Evaluation

We evaluate the performance of the adaptive control with respect to two aspects: *(i)* its robustness with respect to several sources of variations on different clusters (Section V-B), and *(ii)* the benefit of the `two-runs` initialization (Section V-C). The results are consolidated with massive experimental campaigns (over 700 runs; see Section V-D). First, we describe our experimental setup. Emphasis in this evaluation is on representativity of results and experimental reproducibility.

A. Reproducible Experimental Testbed

The experiments presented in this work were conducted on the Grid'5000 testbed [21]. The machines used to run the experiments were selected to contain processors implementing the RAPL mechanism and with varying amounts of RAM and number of sockets. Table II recapitulates the main characteristics of the selected machines.

The experiments were run on a deployed environment: a mechanism offered by Grid'5000 to customize in a reproducible way the whole software stack. A high-level overview of the software stack has been described in Section II-A. We conducted the experiments with the same application used in the previous work [14]: the STREAM benchmark [22]. STREAM was chosen because it is representative of memory-bound phases of applications and shows a stable behavior. STREAM is also easy to modify into an iterative application, which allows computation of the progress metric by reporting heartbeats. The application is instrumented to report a heartbeat to NRM each time the main loop completes an iteration. A thorough description of the whole setup—from environment deployment to how to reproduce the experiments—is available in the Figshare repository of the original work [23].

Given that the system variation caused by the different clusters is well known and mastered, it is dealt with by performing reidentification rather than adaptation. Model parameters per cluster are presented in Table I. In the adaptive control formulation we use (computed from Eqs. (6) and (18))

$$b_0 = \frac{K\Delta_t}{\Delta_t + \tau}. \tag{29}$$

The adaptive approach will allow one to refine the control to the node, run, and runtime variation, as motivated in Section III-B. We use constant reference signals, with values ranging from 0% up to 50% performance degradation ($\epsilon \in [0, 0.5]$) [14]. Note that in our HPC context a degradation over 10% ($\epsilon > 0.1$) is hardly conceivable; and higher values are used to test extreme behavior of our system.

Cluster	CPU	Cores/CPU	Sockets	RAM
`gros`	Intel Xeon Gold 5220	18	1	96
`chifflot`	Intel Xeon Gold 6126	12	2	192
`dahu`	Intel Xeon Gold 6130	16	2	192

TABLE II: Grid'5000 clusters hardware characteristics. RAM quantity is expressed in GiB.

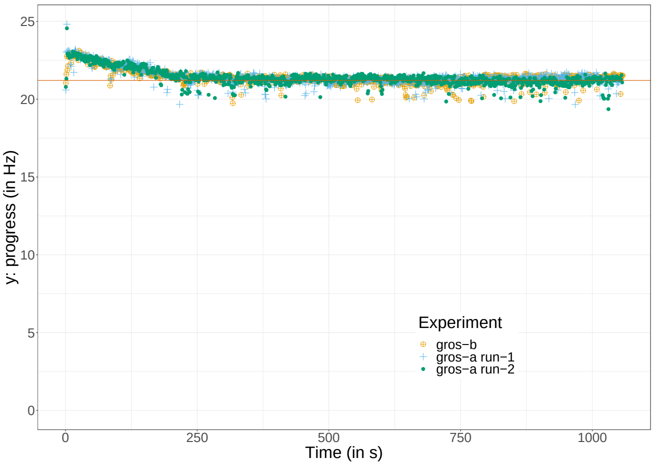

Fig. 8: Adaptive controller performance on the `gros` cluster, for several nodes and runs. Reference is set with $\epsilon = 0.15$ (red continuous line).

B. Robustness Evaluation

We present the performance of our adaptive control for different clusters, nodes, and runs. Comparison can be done with the state-of-the-art PI control (Fig. 5).

Figure 8 presents the output reference tracking of our adaptive controller with `two-runs` initialization for the `gros` cluster. Three experiments are presented, showing results on various nodes and various runs. Figure 8 shows that the control is stable, converging to the reference without static error in about $200\,\mathrm{s}$ and without any oscillations. Note that noise is still present in the measure, with no impact on the stability or the performance. To conclude, the controller is *(i)* efficient in tracking the reference, *(ii)* robust to noise, and *(iii)* robust to node- or run-induced variations. Eventually we evaluate the controlled system w.r.t. our objectives of energy savings; however, graphical representation are omitted due to page limitations. Varying the reference allows one to leverage the energy gains and performance degradation, reducing up to 25% of the energy consumption with only 6% increase in the application execution time.

Figure 9 depicts two runs of the adaptive controller on the `dahu` cluster, chosen to highlight our control evaluation. With respect to the state-of-the-art PI, the adaptive controller is robust to the progress drops and provides a smoother powercap actuation (right plot). We also observe less noise on the progress signal in steady state (left plot). Most important, the system remained stable within the degradation levels of interest (*i.e.*, $\epsilon < 0.1$), which was not the case for all runs of the PI (see Fig. 5, right plot).

Although the system is more stable and robust, such advantages come at the cost of a slower settling time and the presence of an overshoot during the transient state. The PI controller is roughly one order of magnitude faster than the adaptive controller. The progress of the application during the transient state drops about $40\,\%$ with the adaptive controller.

C. Sensitivity to Initialization

We now analyze the benefit of our `two-runs` initialization technique. Performance of the control is presented in Fig. 10 for *1st-run* (left) and *2nd-run* (right). The output reference tracking is plotted, as well as the control signal and adaptive parameter estimate.

Results show that both controllers are stable and converge close to the reference. We note that the initial control action is significantly different in the two cases. For *1st-run*, the control action is initially very low and climbs up until convergence. For *2nd-run*, the control is better initialized, resulting in a smaller performance gap at the execution beginning. The response time with *1st-run* is larger than with a proper initialization (*2nd-run*). Note also that the initial value of progress above the reference is not detrimental for the application performance in our use case; even though it allows for less energy savings.

The parameter estimate $\hat{\theta}$ is adapted throughout the experiments. In *1st-run*, we observe variation until convergence in less than $100\,\mathrm{s}$. The converged value $\theta_\infty^{\mathrm{1st}}$ is a fair estimate, for the rest of the *1st-run* experiment as well as for *2nd-run*.

Discontinuities in the estimate—large drops—are visible, however, in both experiments. They correspond to the presence of outlier measures of progress. The estimation quickly reacts to this newly observed system behavior, while the adaptation of the control action is soundly limited. Despite the presence of outliers and noise, the adaptive control maintains the system stable. Once outlier measures disappear, the estimate quickly converges back to the adequate value (close to $\theta_\infty^{\mathrm{1st}}$). Note that this estimate could be used in this application to detect the presence of outliers—whose exact cause remains uncertain.

Our global objective of energy savings is also visible in Fig. 10. Power around $90\,\mathrm{W}$ is sufficient to run the application with a limited degradation of 15%. Compared with the nominal cluster power of $120\,\mathrm{W}$, it represents 25% savings.

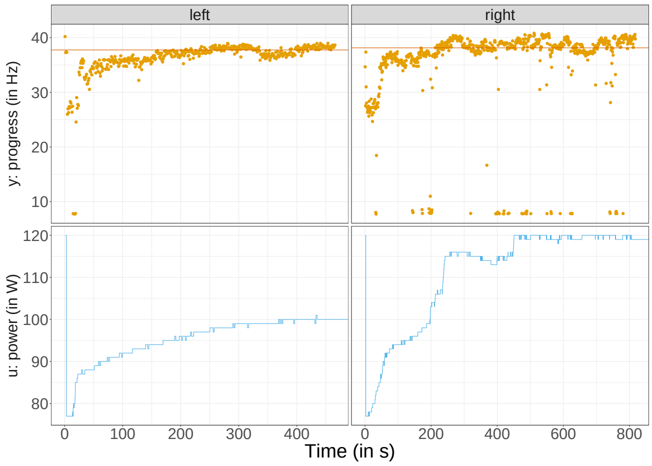

Fig. 9: Adaptive controller performance on the `dahu` cluster, for several runs.

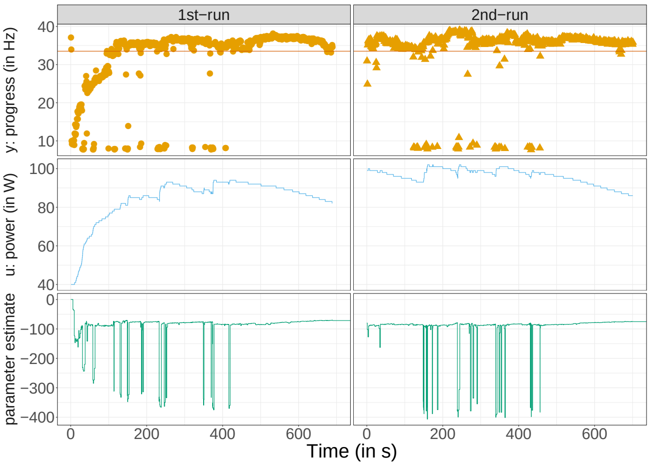

Fig. 10: `two-runs` evaluation: adaptive controller performance y, control signal u, and adaptation parameter estimate θ for the `chifflot` cluster. Reference is set with $\epsilon = 0.15$ (red line). Initialization: *1st-run* (left), *2nd-run* (right).

D. Consolidating Results on Multiple Runs

Results presented so far were illustrated with selected executions of the controller. To provide greater confidence in the results, we have carried out massive campaigns of evaluation—735 runs in total. We present in this section *aggregated* results. While aggregated representations are not common in the control community, we advocate that they promote the realization of solid evaluations and strengthen the results.

Figure 11 shows the distribution of the tracking error per cluster. We first note that all the distributions are centered on 0, reflecting zero static error. For the `gros` cluster, the distribution is narrow since the convergence time is short and the progress measure has little noise. For the `dahu` cluster, we observe a wider distribution since the measure is subject to much more noise. This representation is also interesting because it allows one to quantify the noise—here of $5\,\mathrm{Hz}$ when referring to quartiles. The asymmetry of the distribution reveals the presence of overshoot (around

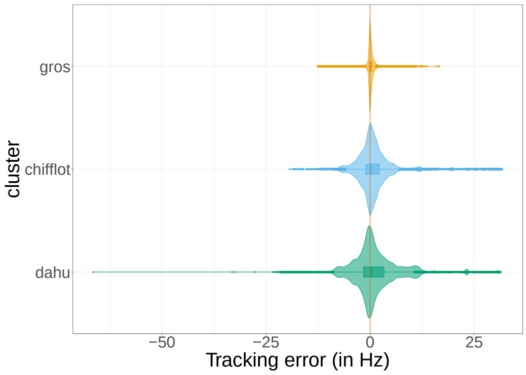

Fig. 11: Aggregation of tracking error over multiple adaptive control repetitions on different nodes for each cluster (735 runs in total), *2nd-run*.

10 Hz) and exhibits outliers (spread around 25 Hz). The `chifflot` cluster has an error distribution close to that of `dahu`, since they are both dual-socket clusters. The static noise is smaller (narrower distribution) and the overshoot limited (almost symmetrical distribution). The large tails for `dahu` and `chifflot` may be tackled by a better parameter estimation. Initialization using more runs is considered as future work.

VI. Conclusion

In this work we aimed at minimizing energy consumption while maximizing the performance of applications in HPC systems. We motivated the problem of power regulation in computing systems along with the use of control theory in this domain. We formulated the problem to be tackled based on existing work using a PI controller design. We then discussed the limitations of this approach on the model and control sides. Our aim was to use a controller that is robust to different clusters and that does not rely on numerous parameters computed offline. We proposed using a new controller based on adaptive control. In terms of controller design, this allows one not to model the nonlinear behaviors of the system and reduces the number of model parameters from six down to one. The adaptation relies on Model-Reference Adaptive Control for which we designed `two-runs`, a novel initialization process. We evaluated the controller on a real system, namely, various clusters of the Grid'5000 testbed. The experimental results show significant improvements over the previous PI controller on stability and robustness. In addition to the classical advantages of adaptive control, our solution is robust to variations of the machines (from one node to another) and of the runs (from one execution of the application to another). These dimensions of robustness are less usual advantages of adaptive control. Such a controller design improves reusability and simplicity: two important attributes from a computer science point of view. Overall, we were able to further reduce the energy consumption, up to 25% savings for the single-socket cluster. The results of the adaptive control are better than those of existing PI control, and present interesting possibilities for future work:

- Adaptive control speed: We noticed that the controller needs 200 s to converge to the required level, which could be improved.
- Evaluation of the approach for different applications: Here we evaluated only for STREAM, which is a memory-bound application.
- Addition of sensor: A sensor such as a temperature sensor might be useful in trying to identify the source of disturbances.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations (see https://www.grid5000.fr). Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computer Research, under Contract DE-AC02-06CH11357. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was partially supported by JLESC: the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint-Laboratory for Extreme Scale Computing (https://jlesc.github.io/).

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, pp. 41–50, Jan. 2003.

[2] É. Rutten *et al.*, "Feedback Control as MAPE-K Loop in Autonomic Computing," in *Software Engineering for Self-Adaptive Systems*, vol. 9640 of *Lecture Notes in Computer Science*, pp. 349–373, Springer, 2013.

[3] J. L. Hellerstein *et al.*, *Feedback Control of Computing Systems*. Wiley, 2004.

[4] A. Filieri *et al.*, "Software Engineering Meets Control Theory," in *SEAMS@ICSE*, pp. 71–82, IEEE, May 2015.

[5] M. Berekmeri *et al.*, "Feedback Autonomic Provisioning for Guaranteeing Performance in MapReduce Systems," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1004–1016, 2018.

[6] T. Nylander *et al.*, "Cloud Application Predictability through Integrated Load-Balancing and Service Time Control," in *ICAC*, pp. 51–60, IEEE, 2018.

[7] A. V. Papadopoulos *et al.*, "Hard Real-Time Guarantees in Feedback-Based Resource Reservations," *Real-Time Syst.*, vol. 51, June 2015.

[8] Q. Guilloteau *et al.*, "Model-Free Control for Resource Harvesting in Computing Grids," in *CCTA 2022 - 6th IEEE Conference on Control Technology and Applications*, IEEE, Aug. 2022.

[9] A. G. Yabo *et al.*, "A control-theory approach for cluster autonomic management: maximizing Usage While Avoiding Overload," in *CCTA 2019 - 3rd IEEE Conference on Control Technology and Applications*, (Hong Kong, China), pp. 189–195, IEEE, Aug. 2019.

[10] C. Imes *et al.*, "POET: A Portable Approach to Minimizing Energy Under Soft Real-time Constraints," in *RTAS*, pp. 75–86, IEEE, 2015.

[11] E. Rotem *et al.*, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[12] C. Imes *et al.*, "CoPPer: Soft Real-time Application Performance Using Hardware Power Capping," in *ICAC*, pp. 31–41, IEEE, 2019.

[13] S. Ramesh *et al.*, "Understanding the Impact of Dynamic Power Capping on Application Progress," in *IPDPS*, pp. 793–804, 2019.

[14] S. Cerf *et al.*, "Sustaining Performance While Reducing Energy Consumption: A Control Theory Approach," in *Euro-Par*, vol. 12820 of *Lecture Notes in Computer Science*, pp. 334–349, Springer, 2021.

[15] V. Reis *et al.*, "Argo Node Resource Manager." https://www.mcs.anl.gov/research/projects/argo/overview/nrm/, 2021.

[16] S. Desrochers, C. Paradis, and V. M. Weaver, "A Validation of DRAM RAPL Power Measurements," in *MEMSYS*, pp. 455–470, ACM, 2016.

[17] S. Akhtar and D. S. Bernstein, "Lyapunov-stable discrete-time model reference adaptive control," *Int. J. Adapt. Control Signal Process.*, vol. 19, no. 10, pp. 745–767, 2005.

[18] W. S. Black *et al.*, "Adaptive Systems: History, Techniques, Problems, and Perspectives," *Systems*, vol. 2, no. 4, pp. 606–660, 2014.

[19] C. Rohrs *et al.*, "Robustness of Continuous-Time Adaptive Control Algorithms in the Presence of Unmodeled Dynamics," *IEEE Transactions on Automatic Control*, vol. 30, no. 9, pp. 881–889, 1985.

[20] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*. Courier Corporation, 2014.

[21] D. Balouek *et al.*, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *CLOSER (Selected Papers)*, vol. 367 of *Communications in Computer and Information Science*, pp. 3–20, Springer, 2012.

[22] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, 1995.

[23] S. Cerf *et al.*, "Artifact and Instructions to Generate Experimental Results for the Euro-Par 2021 Paper: "Sustaining Performance While Reducing Energy Consumption: A Control Theory Approach"." https://doi.org/10.6084/m9.figshare.14754468, Aug. 2021.